CSE 378 Machine Architecture and Assembly Language

PRACTICE QUESTIONS:

1) Programming in SPIM. Write a program in MIPS assembly language that finds (i)the number of elements strictly greater than the last element,(ii) the minimum element,(iii) the maximum element, and (iii) the (integer) average of all elements in an array of integers (each element is a 32-bit positive, negative, or null integer). Your input, the array and its size, should be in ".data" section of your program. As output, your program should display the 4 values asked for above on the console with appropriate messages.

2) Implement the following in MIPS:

(i) forloop.c - A for loop

```
int forloop(){

        int i = 0;
        int result = 0;
        for (i; i<20; i=i+2){
                result += i;
        }
        return result;

}
```

(ii) funccall.c - A function calling example

```
int main(){

        int a = 4;

        return helper(a);


}

int helper(int a){

        int b = a * 4;
        return b;

}
```

(iii) strcat.c - C string concatenation function

```
char* strcat (char * dst, char * src){

        char* originalDst = dst;

        while(*dst){
                dst++;
        }

        while(*src){
```

```
            *dst = *src;
            dst++;
            src++;
        }

        *dst = *src;

        return originalDst;

}
```

3) <§5.5> Show how the jump register instruction (described on pages 76 and A-62) can be implemented simply by making changes to the finite-state machine of Figure 5.38 on page 359. (It may help you to remember that $0 = $zero = 0.)

4)<§5.4> We wish to add the instruction addi (add immediate) to the single-cycle datapath described in this chapter. Add any necessary datapaths and control signals to the single-cycle datapath of Figure 5.17 on page 307 and show the necessary additions to Figure 5.18 on page 308. You can photocopy these figures to make it faster to show the additions.

5) <§5.4> This question is similar to Exercise 5.19 except that we wish to add the instruction jal (jump and link), which is described in Chapter 2 on page 79. You may find it easier to modify the datapath in Figure 5.25 on page 318.

6) <§5.4> This question is similar to Exercise 5.19 except that we wish to add a variant of the lw (load word) instruction, which sums two registers to obtain the address of the data to be loaded and uses the R-format.

7) <§5.4> Explain why it is not possible to modify the single-cycle implementation to implement the swap instruction described in **In More Depth:** Instruction Set Styles, Exercise 2.52, without modifying the register file.

8) <§5.5>We wish to add the instruction lui (load upper immediate) described in Chapter 3 to the multi-cycle data-path described in this chapter. Use the same structure of the multi cycle data-path of Figure 5.28 on page 323 and show the necessary modifications to the finite-state machine of Figure 5.37 on page 338. You may find it helpful to examine the execution steps on pages 325 through 329 and consider the steps that will need to be performed to execute the new instruction. How many cycles are required to implement this instruction?

9) <§6.1>Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding? Which dependencies are data hazards that will cause a stall?

```
add     $3, $4, $2
sub     $5, $3, $1
lw      $6, 200($3)
add     $7, $3, $6
```

10) <§§6.4,6.5> Consider executing the following code on the pipelined data-path of Figure 6.36 on page 416:

```
lw      $4, 100($2)
sub     $6, $4, $3
add     $2, $3, $5
```

How many cycles will it take to execute this code? Draw a diagram like that of figure 6.34 on page 414 that illustrates the dependencies that need to be resolved, and provide another diagram like that of Figure 6.35 on page 415 that illustrates how the code will actually be executed (incorporating any stalls or forwarding) so as to resolve the identified problems.